Целые числа. Битовые операции

Шокуров Антон В.
shokurov.anton.v@yandex.ru
http://машинноезрение.рф

16 февраля 2017 г.

Версия: 0.12

Аннотация

Целые числа имеют свою специфику. Используются в целочисленной арифметике, а также для вычисления битовых операций.

Цель. Изучить представление целых чисел. В частности, битовые операции над ними.

Предварительный вариант!

1 Целые числа

Как отмечалось уже в самой первой заметке целые числа имеют свою специфику, в частности, специальные вычислительные операции над ними.

1.1 Представление

Что есть целое число рассматривается в курсе алгебры. Оно в частности не имеет дробной части.

Десятичное представление. Неотрицательные целые числа можно записать в десятичной системе записи. Необходимо четко понимать, что фактически это означает следующее:

$$123 \sim 1 * 100 + 2 * 10 + 3 * 1 \sim 1 * 10^2 + 2 * 10^1 + 3 * 10^0.$$

Таким образом число представляется в виде суммы степеней числа десять (10^i) умноженных на некий неотрицательный коэффициент (a_i) меньшей 10: $\sum_{i=0}^{i=k} a_k 10^k$,

где $a_k \neq 0, 0 \leq a_i < 10$. Коэффициент a_k является старшим, точнее говоря, можно сказать, что для представления числа потребовалось $k+1$ разрядов. Само число можно писать в виде 123_{10}, где цифра 10 указывается для того, чтобы подчеркнуть десятичную систему исчисления.

Такое разложение можно выполнить не только для числа 10, но и для других положительных целых чисел, т.е. по других основаниям. В других системах исчислениях дабы это подчеркнуть в качестве правого нижнего индекса указывается соответствующее число.

С вычислительными системами тесно связаны двоичная и шестнадцатеричная система исчисления.

Двоичное представление Можно представить число и в родной компьютеру системе исчисления, а именно – в *двоичной* системе (иначе, *бинарное представление*). В данной системе число записывается не по базе 10, а по базе 2, т.е. i разряд соответствует числу 2^i. Соответственно a_i будут просто равны либо 0 либо 1. Для рассмотренного ранее числа:

$$123_{10} \sim 64 + (123-64) \sim 64 + 59 \sim 64 + 32 + (59-32) \sim 64 + 32 + 27 \sim$$

$$\sim 64 + 32 + 16 + (27-16) \sim 64 + 32 + 16 + 11 \sim 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 \sim$$

$$\sim 1*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0.$$

По аналогии с десятичной системой это число можно записать компактно как:

$$123_{10} \sim 1111011_2.$$

Упражнение. Напишите программу, которая печатает двоичное представление введенного числа.

В программах на Си числа в бинарном представлении можно использовать только в последних редакциях (`C++14`):

```
1 int a = 0b1111011;
2 printf("%d\n", a + 0b101);
```

Шестнадцатеричные числа Одна из причин того, что бинарное представление в языке Си отсутствовало так долго, заключается в том, что оно крайне длинное. Более профессионально использовать шестнадцатеричное представление чисел. Продолжая представления того же числа по аналогии для основания 16:

$$123_{10} \sim 7*16^1 + (123 - 7*16^1) \sim 7*16^1 + 11*11^0 \sim (7)(11)_{16}.$$

Для удобства в шестнадцатеричной системе разряды имеющие значение больше равное чем 10 заменяются на буквенное представление:

значение	обозначение	значение	обозначение
10	A	13	D
11	B	14	E
12	C	15	F

В согласие с отмеченными обозначениями:

$$123_{10}\sim(7)(11)_{16}\sim 7b_{16}.$$

В программах на Си числа в шестнадцатеричном представлении можно записать следующим образом:

```
1 int a = 0x7b;//В качестве префикса используется
2 //комбинация 0x или 0X.
3 printf("%d\n", a + 0x11);
```

Более того, в отличии от двоичного представления числа можно напечатать в шестнадцатеричном представлении используя нужный формат в printf

```
1 int a = 123;
2 //Шестнадцатеричное представление при печати printfom
3 //задается буквой x или X:
4 printf("%x\n", a);//x - печатает все буквы строчные.
5 //Напечатает: 7b
6 printf("%X\n", a);//x - печатает все буквы заглавными.
7 //Напечатает: 7B
```

По аналогии с печатью числа могут быть считаны в шестнадцатеричной системе исчисления:

```
1 int a;
2 //Шестнадцатеричное представление при печати printfom
3 //задается буквой x или X:
4 scanf("%x", &a);//x - регистр буквы не важен.
5 printf("in dec: %d\n", a);
6 //Напечатает введенное в шестнадцатеричной системе число
7 //в десятичной системе.
```

При записи чисел альтернативой к $7b_{16}$ является $7bh$, т.е. в конце числа добавляется символ h. Это связано с тем, что не всегда есть возможность напечатать нижний индекс, в особенности это касается число текстовых файлов, например, исходных файлов с программой.

Упражнение. Доделайте прошлый фрагмент кода так, чтобы он вывел число как 7bh.

Упражнение. Напишите программу которая выводит число в произвольной системе исчисления.

Восьмеричные числа Для восьмеричной системы справедливо:

$$123\sim123_{10}\sim173_8.$$

В программах на Си числа в восьмеричном представлении можно записать следующим образом:

```
1 int a = 0173;//В качестве префикса используется цифра 0.
2 printf("%o\n", a + 011);
```

Печать и считывание в восьмеричной системе полностью аналогична шестнадцатеричной системе. В формате вместо x нужно использовать o:

```
1 int a;
2 //Восьмеричное представление при печати printfom
3 //задается буквой o:
4 scanf("%x", &a);
5 printf("in oct: %o\n", a);
6 //Напечатает введенное в шестнадцатеричной системе число
7 //в восьмеричной системе.
```

1.2 Целые числа в Си

Как было показано в предыдущем подразделе число можно представить в двоичной системе исчисления. Каждый разряд может быть либо 0 либо 1, минимальный элемент памяти – *бит*, который фактически соответствует неким физическим процессам. В памяти компьютера можно хранить только ограниченное количество битов. С числами рациональное оперировать когда они имеют одинаковое количество битов, т.е. представлены фиксированным заранее определенным количеством битов.

В следствии чего в компьютере целые числа имеют несколько подтипов, характеризующиеся количеством разрядов в представлении числа. С точки зрения удобства количества битов в представление числа увеличивается в два раза с каждым увеличением разрядности числа. Стандартно: 8, 16, 32, 64 и так далее.

Знаковый тип Данный тип данных уже не раз ранее встречался в заметках:

```
1 //Объявили переменную типа int, целого, знакового.
2 int a;
```

Есть типы встроенные в сам язык: `char`, `short`, `int`, `long int`, `long long int`. Об их размере известно только то, что он не убывает.

Такое представление Кольцо вычетов.

Без знаковый Добавляется слово unsigned:

```
1 //Объявили переменную типа unsigned int,
2 //целого, беззнакового.
3 unsigned int a;
```

Такую модификацию можно сделать с любым из целым типом ранее перечисленных: `unsigned char`, `unsigned short`, `unsigned int`, `unsigned long int`, `unsigned long long int`.

1.3 Платформонезависимые типы

Рассмотренные выше типы языка Си (char, short, int, ...) лучше всего использовать в обычных вычислениях, где не важен точный битовый размер каждого из типов данных. В задачах, где как раз битовый размер важен, предпочтительно использовать предопределенные специальные типы. Отмечу, что рассматриваемые вспомогательные типы данных не задаются языком Си, а являются частью часто идущим с ним стандартной библиотеки (они определяются в заголовочном файле `stdint.h`).

Объявление переменных Для создания типа нужного размера лучше воспользоваться вспомогательными знаковыми типами: `int8_t`, `int16_t`, `int32_t` и тому подобные. Они гарантируют нужный размер в битах. Соответственно вспомогательные беззнаковые типы обозначаются как: `uint8_t`, `uint16_t`, `uint32_t` и тому подобные.

Ввод и вывод Для выполнения операций ввода/вывода лучше всего использовать опять же корректный способ. Для это придется добавить ещё один заголовочный файл `inttypes.h`. Тогда

```
1 //Объявили переменную типа int, целого, знакового.
2 uint16_t a;//16 бит, беззнаковое
3 //что соответствует формату чтения SCNu16:
4 scanf("%" SCNu16 "\n", &a);
5 int32_t b = a +10;//32 бита, знаковый
6 //что соответствует формату печати PRIi32:
7 printf("%" PRIi32 "\n", b);
```

Этих вариантов много. В большинстве случаев можно догадаться, иначе можно просто открыть отмеченный заголовочный файл и проверить.

1.4 Двоичные операции

Считается что число имеет фиксированное количество битов в его представлении. Битовые операции выполняются по-битово, т.е. для каждого из разрядов по-отдельности. Последнее фактически означает, что результат операции для какого либо разряда не влияет на результат действия двоичной операции на другие разряды. Можно считать, что операции выполняются в неком смысле в параллель.

Унитарная операция Выполняется с одним битом одного операнда (числа). Инвертирование каждого из битов, в языке Си обозначается операцией ~:

бит	~
0	1
1	0

Так,

$$(\sim 011001_2)\sim(100110_2).$$

В последнем выражении просьба не путать унитарную операцию и операцию сведения к числу.

Бинарные операции Действуют между двумя числами. В Си заданы следующие бинарные битовые операции: и (см. а), или (см. б.) и либо (см. в.).

а)

&	0	1
0	0	0
1	0	1

б)

|	0	1
0	0	1
1	1	1

в)

∧	0	1
0	0	1
1	1	0

Не путать с логическими операциями И и ИЛИ в которых используется двойной символ (соответственно, && и ||).

```
1 //Объявили переменную типа int, целого, знакового.
2 int a=0b00001101;
3 int b=0b10101010;
4 int c=a&b;
5 //c~0b00001000
6 int d=a|b;
7 //d~0b10101111
8 int e=a^b;
9 //e~0b10100111
```

Битовый сдвиг С точки зрения битового представления в процессорах есть и в языке Си реализована операция побитового сдвига, а именно – число можно по-битово сдвинуть как влево(<<), так и право (>>). По сути этим операциям соответствует умножение и деление на 2. Но есть тонкости.

```
1 //Объявили переменную типа int, целого, знакового.
2 int a=0b00001101;
3 int b=a<<1;
4 //b~0b00011010
5 int a=0b00001101;
6 int c=a>>1;
7 //c~0b00000110
```